RLSRunner: Linking Rascal with K for Program Analysis

Mark Hills, Paul Klint, & Jurgen J. Vinju

4th International Conference on Software Language Engineering
July 4, 2011
Braga, Portugal

Some Quick Definitions

- Rascal: A meta-programming language for source code analysis and transformation

- K: A semantic framework, based on concepts from rewriting logic (a logic of concurrency) and term rewriting

- Maude: A language and execution engine for rewriting logic specifications

Overview

Overview

- Motivation

Overview

- Motivation
- Tool components: Rascal

Overview

- Motivation
- Tool components: Rascal
- Tool components: K

Overview

- Motivation
- Tool components: Rascal
- Tool components: K
- Demo

Overview

- Motivation
- Tool components: Rascal
- Tool components: K
- Demo
- Wrap-up

Motivation: Why This Tool?

Motivation: Why This Tool?

- Many K and K in Maude specifications exist -- want reuse

Motivation: Why This Tool?

- Many K and K in Maude specifications exist -- want reuse
- Integrating with graphical environments currently ad-hoc, bad user experience

Motivation: Why This Tool?

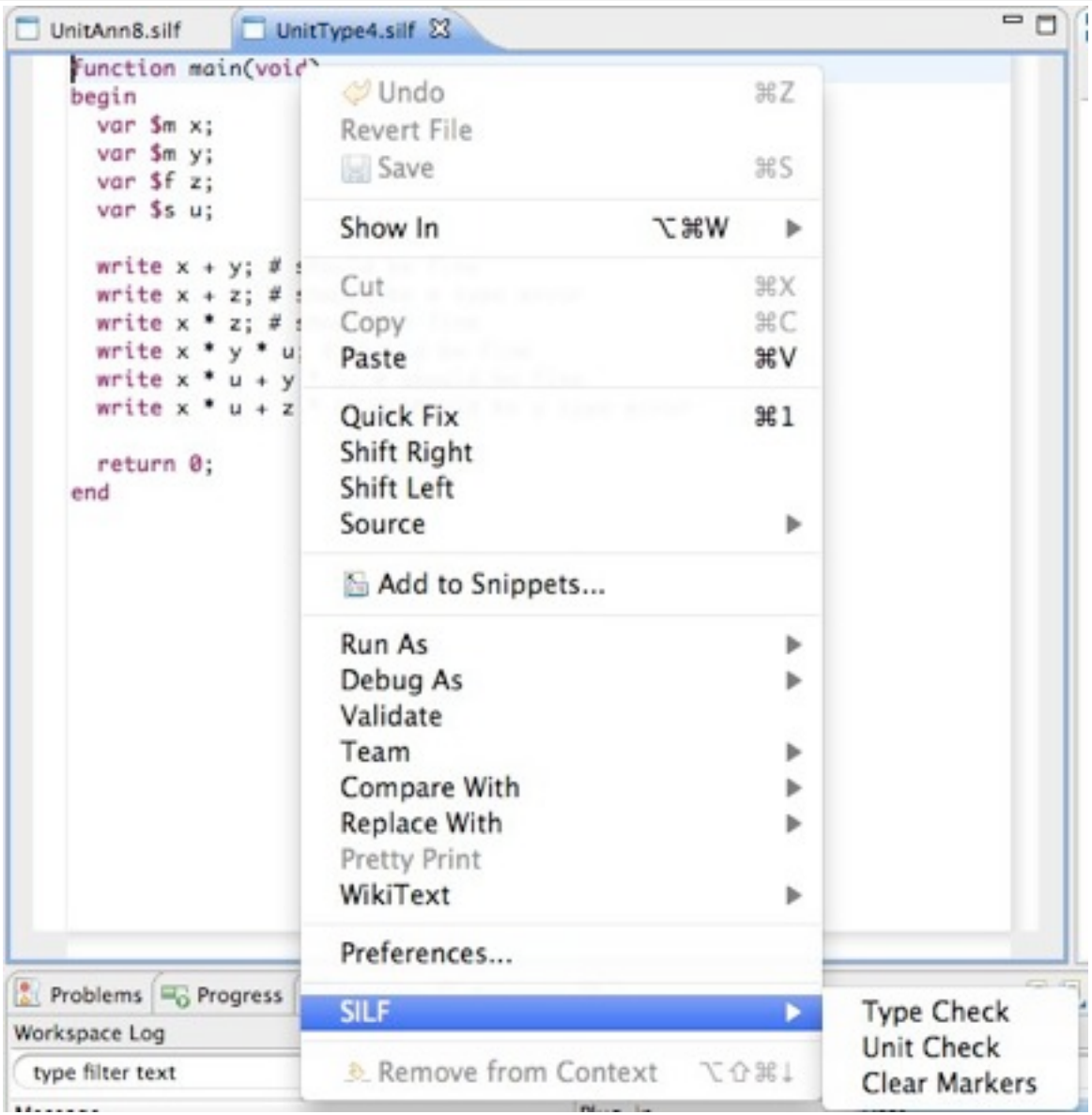
- Many K and K in Maude specifications exist -- want reuse
- Integrating with graphical environments currently ad-hoc, bad user experience
- Want a general method to integrate these specifications with Rascal-based IDEs

Motivation: Why This Tool?

- Many K and K in Maude specifications exist -- want reuse
- Integrating with graphical environments currently ad-hoc, bad user experience
- Want a general method to integrate these specifications with Rascal-based IDEs
- (Personal) Wanted something like this all during my PhD

How Should it Work?

How Should it Work?



How Should it Work?

How Should it Work?

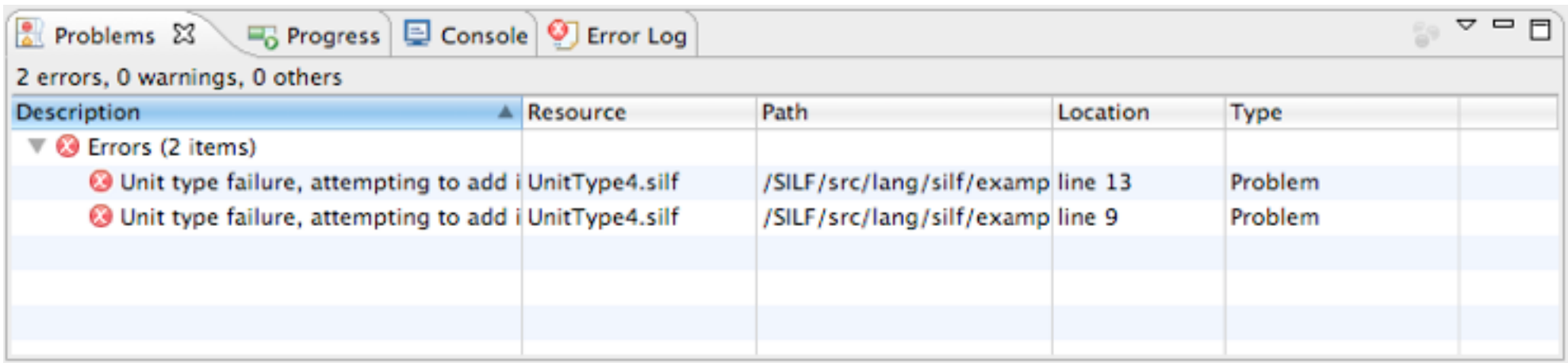
```
function main(void)
begin
  var $m x;
  var $m y;
  var $f z;
  var $s u;

  write x + y; # should be fine
  write x + z; # should be a type error
  write x * z; # should be fine
  write x * y * u; # should be fine
  write x * u + y * u; # should be fine
  write x * u + z * u; # should be a type error

  return 0;
end
```

How Should it Work?

How Should it Work?



Tool Components: Rascal (ShellExec)

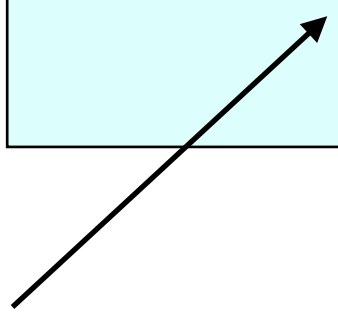
Rascal

```
PID pid = createProcess(maudeLocation.path);
writeTo(pid, toRun);
res = readFrom(pid);
killProcess(pid);
```

Tool Components: Rascal (ResourceMarkers)

Rascal

```
data Message = error(str msg, loc at)
             | warning(str msg, loc at)
             | info(str msg, loc at);
```



Rascal

```
import Message;

public void java removeMessageMarkers(loc resourceLoc);

public void java addMessageMarkers(set[Message] markers);
```

Tool Components: Rascal (RLSRunner ADT)

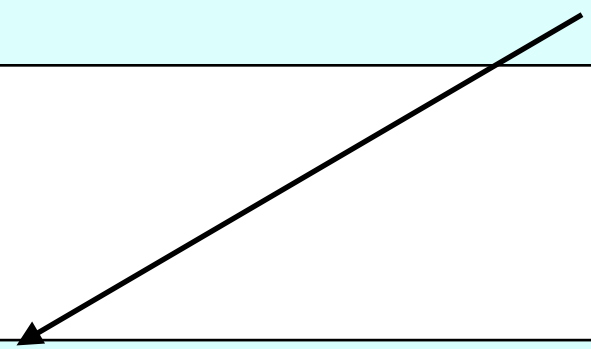
Rascal

```
data RLSRunner = RLSRun(loc maudeFile,
                        str(str,list[str]) pre,
                        RLSResult(str) post);
```

Tool Components: Rascal (Maude-ifier)

```
if ((Program)`<Decl* decls> <FunDecl+ funDecls>` := p)
   return located(p,"Pgm",
                  "__(<showDecls([d|d<-decls])>,
                      <showFunDecls({f|f<-funDecls})>)");
```

Rascal



```
syntax Program = Default: Decl* decls FunDecl+ funDecls;
```

Rascal

Tool Components: Rascal (Returning Results)

Rascal

```
data RLSResult = SILFAnalysisResult(bool foundErrors,
                                    set[Message] messages) ;
```

Rascal

```
void exec(Tree pt, loc l) {
   str pgm = maudeify(pt, true, policy);
   RLSRunner rlsRunner = RLSRun(silfSpec, pre, post);
   RLSResult res = runRLSTask(maudeExec, rlsRunner, pgm);
   if (SILFAnalysisResult(true,msgs) := res)
      addMessageMarkers(msgs);
}
```

Tool Components: K (Rascal Source Locations)

K/Maude

```
fmod RASCAL-LOCATION is
  including STRING .
  including INT .
  sort RLocation .
  op sl : String Int Int Int Int Int Int -> RLocation .
endfm
```

Tool Components: K (Location Semantics)

K/Maude

```
op currLoc : RLocation -> State [format (r! o)] .

op rloc : RLocation -> ComputationItem .

eq k(rloc(RL) -> K) currLoc(RL') = k(K) currLoc(RL) .

eq k(exp(locatedExp(E, RL)) -> K) currLoc(RL') =
     k(exp(E) -> rloc(RL') -> K) currLoc(RL) .
```

Tool Components: K (Generating Results)

K/Maude

```
op makeAnalysisMsg : OutputList -> String .

eq makeAnalysisMsg(warning(level(1) msgloc(RL) msg(S) WIS), OL) =
    ("||1:::" + rloc2str(RL) + ":::" + S + "||") + makeAnalysisMsg(OL) .
```

Demo

Future Work

- Support other execution features: standard execution, model checking, state space search
- Automatic generation of Maude-ifier and Maude operator defs for abstract syntax
- Support C Policy Framework

Wrap-Up

- RLSRunner provides a reusable library for running Maude-based K definitions
- Per-language requirements light (outside of maude-ifier), leverage library
- K specifications require minimal modifications
- Provides a template for integrating other console apps with Rascal